head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 97.01.24.01.11.19; author airey; state Exp; branches; next ; desc @@ 1.1 log @shader pages @ text @

MAPPING RENDERMAN TO OPENGL

The RenderMan data types can be mapped onto data in the OpenGL pipeline as follows: float is either a single component buffer or the red channel of a color buffer; point and color are three-component vectors that map to the red, green, and blue channels of colors in the pipe. Strings are kept on the host.

```
float       scalar variable (no other scalar data types)
string       file identifier
point       vector of three floats
color       vector of any number (usually 3) of floats
```

Global variables typically can be computed in a single pass from the host. The most touchy parameters are the surface derivatives. We do need a pass-through of the interpolated and normalized vectors (such as the normal vector) and a gen-color mechanism to get interpolated points (such as P).

```
color   Cs      Surface color (input)
color   Os      Surface opacity (input)
point   P       Surface Position
point   dPdu    Change in position with 'u'
point   dPdv    Change in position with 'v'
point   N       Surface shading normal
point   Ng      Surface geometric normal
float   u,v     Surface parameters
float   du,dv   change in u,v across element
float   s,t     surface texture coordinates
float   L       Direction from surface to light source
color   Cl      Light color
point   Ci      Color of light from surface (output)
point   Oi      Opacity of surface (output)
point   E       Position of the camera
point   I       Direction of ray impinging on surface
```

The RenderMan operators map relatively painlessly to the operations already present in OpenGL. In some cases, however, we will need to use a lookup table to achieve the results. We often have several choices of OpenGL operations to reach our goals; the decision made will depend on the best option for a given platform.

```
    ()                  left        expression grouping
    -!                  right        unary arithmetic and logical negation
    .                  left        dot product
    */                  left        multiplication and division
    ^                  left        cross product
    +-                  left        addition and subtraction
    = >         left        arithmetic comparison
    == !=                   left        equal and not equal
    &&                  left        logical and
    ||                  left        logical or
    ?:                  right        conditional expression
    =                  right        assignment
```

Arithmetic (a and b are float, point, or color):

```
    a = b

    Copy(b);

    -a

    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
glColor4f(-1.,-1.,-1.,1.);
```

head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 97.01.24.01.11.19; author airey; state Exp; branches; next ; desc @@ 1.1 log @shader pages @ text @

```
    Draw(object);

    a + b

    Draw(a);
glBlendFunc(GL_ONE,GL_ONE);
    Draw(b);

    a - b

    Draw(a);
glBlendFunc(GL_ONE,GL_ONE);
glBlendEquationEXT(GL_FUNC_SUBTRACT_EXT);
    Draw(b);

    a * b

    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
    Draw(b);

    a / b

    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
glEnable(GL_PIXEL_TEXTURE);
glTexImage1D(inverse table);
    Copy(b);
```

Vector Operations (a and b are points):

```
    a . b (dot product)

    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
    Draw(b);
    Set ColorMatrix
    Copy(result);

    a ^ b (cross product)

    /* a^b = ( (ya*zb-za*yb) (za*xb-xa*zb) (xa*yb-ya*xb) )
           = (ya za xa)*(zb xb yb) - (za xa ya)*(yb zb xb)
           = ( (ya za xa)*(zb xb yb)/(yb zb xb) - (za xa ya) ) * (yb zb xb) */
    Set ColorMatrix
    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
    Set ColorMatrix
    Draw(b);
glEnable(GL_PIXEL_TEXTURE);
glTexImage1D(inverse table);
    Set ColorMatrix
    Draw(b);
glDisable(GL_PIXEL_TEXTURE);
glBlendEquationEXT(GL_FUNC_SUBTRACT_EXT);
glBlendFunc(GL_ONE,GL_ONE);
    Set ColorMatrix
    Draw(a);
glBlendEquationEXT(GL_FUNC_ADD_EXT);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
    Set ColorMatrix
    Draw(b);
```

Logical Operations (a and b are float, color, or point):

```
    a == b

    Draw(a);
glBlendEquationEXT(GL_FUNC_EQUAL_EXT);
```

head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 97.01.24.01.11.19; author airey; state Exp;
branches; next ; desc @@ 1.1 log @shader pages @ text @

```
    a != b

    Draw(a);
glBlendEquationEXT(GL_FUNC_NOTEQUAL_EXT);
    Draw(b);
```

Logical Operations (a and b are float):

```
    a glBlendEquationEXT(GL_FUNC_LESS_EXT);
    Draw(b);

    a glBlendEquationEXT(GL_FUNC_LESSEQ_EXT);
    Draw(b);

    a > b

    Draw(a);
glBlendEquationEXT(GL_FUNC_GREATER_EXT);
    Draw(b);

    a >= b

    Draw(a);
glBlendEquationEXT(GL_FUNC_GREATEREQ_EXT);
    Draw(b);
```

Boolean Operations (a and b are boolean):

```
    ! a

glEnable(GL_PIXEL_TEXTURE);
glTexImage1D(not table);
    Copy(a);

    a && b

    Draw(a);
glBlendFunc(GL_DST_COLOR,GL_ZERO);
    Draw(b);

    a || b

    Draw(a);
glBlendEquationEXT(GL_LOGIC_OP);
glLogicOp(GL_OR);
    Draw(b);
```

Mathematical functions all take and return type **float** (c is a constant):

```
sin(a) asin(a)
cos(a) acos(a)
tan(a) atan(a)
radians(a) degrees(a)
sqrt(a) pow(a,c)
exp(a) log(a)
mod(a,c) abs(a)
sign(a) clamp(a,c,c)
ceil(a) floor(a)
round(a) step(c, a)
smoothstep(c, c, a)
```

All monadic functions can be implemented via pixel-texture (c is constant):

```
glEnable(GL_PIXEL_TEXTURE);
glTexImage1D(bltin table);
    Copy(a);
```

head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 97.01.24.01.11.19; author aicey; state Exp; branches; next ; desc @@ 1.1 log @shader pages @ text @

That leaves a handful of other math functions. All but the power function can be handled with the previous calls:

```
step(a, x) {
        return ( (float)(x&gt=a) )
    }

smoothstep(min, max, val) {
        if( x&lta )
            return 0;
        if( x&gt=b )
            return 1;
        x = (x-a)/(b-a);
        return( x*x*(3-2*x) );
    }

clamp(x,a,b) {
        return (x&lta ? a : (x&gtb ? b : x));
    }

min(a,b) {
        return (a&ltb ? a : b);
    }

max(a,b) {
        return (a&ltb ? b : a);
    }

mod(a,b) {
        float t = a/b;
        return ( t-floor(t) );
    }

atan(y,x) {
        if( x>=0. )
            return ( atan(y/x) );
        if( y>=0. )
            return( PI-atan(y/x) );
        return( -PI-atan(y/x) );
    }

pow(x,y) {
        unknown?
    }
```

Finally, RenderMan has a number of other built-in functions. These include:

```
float area(point P) {
        /* texture lod */
    }

point calculatenormal(point P) {
        return( Du(P)^Dv(P) );
    }

float depth(point P) {
        point p = P-I;
        return( sqrt(p.p) );
    }

float distance(point p1, point p2) {
        point p = p2-p1;
        return( sqrt(p.p) );
    }

Deriv(num, denom) or
Du(expr) or
Dv(expr) {
    }
```

head 1.1; access; symbols; locks; strict; comment @# @;   1.1 date 97.01.24.01.11.19; author airey; state Exp; branches; next ;   desc @@   1.1 log @Shader pages @ text @

```
point faceforward(point V, point R) {
        if( V.Rfloat length(point p) {
        return( sqrt(p.p) );
    }

point normalize(point p) {
        return( p/length(p) );
    }

color mix(color c0, color c1, float a) {
        Draw(c0);
glColorMask(0,0,0,1);
        Set ColorMatrix
        Draw(a);
glColorMask(1,1,1,1);
        Draw(c1);
    }

noise(float val) or
noise(float u, floatv) {
        1D or 2D Bicubic Textures
    }

noise(point p) {
        3D Textures or brute force 2D Texture+Lookup
    }

color ambient(void) {
        return( ambient light );
    }

color diffuse(point N) {
        color c = 0;

        for( i=0; i&ltnlights; i++ ) {
            c += Attenuationi*(N.Li);
        }

         return( c );
    }

color phong(point N, point eye, float rough) {
        color c = 0;
        point r;

        r = 2*(N.eye)*N-eye;
        for( i=0; i&ltnlights; i++ ) {
            c += Attenuationi*pow((r.Li),1./rough);
        }

         return( c );
    }

color specular(point N, point eye, float rough) {
        color c = 0;
        point h;

        for( i=0; i&ltnlights; i++ ) {
            h = normalize(Li+eye);
            c += Attenuationi*pow((N.h),1./rough);
        }

         return( c );
    }

setcomp(color a, float index, float b) or
setxcomp(point a, float b) or
setycomp(point a, float b) or
setzcomp(point a, float b) {
```

head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 97.01.24.01.11.19; author airey; state Exp;
branches; next ; desc @@ 1.1 log @shader pages @ text @

```
                Set ColorMatrix
                Copy(b);
            }

        float comp(color c, float index) or
        float xcomp(point a) or
        float ycomp(point a) or
        float zcomp(point a) {
                Set ColorMatrix
                Draw(a);
            }

        spline(a, f1, f2, ...) {
                /* if f1, f2, ... uniform */
        glEnable(GL_PIXEL_TEXTURE);
        glTexImage1D(spline table);
                Copy(a);
                /* else unknown? */
            }

        point bump(string name, point norm, dPds, dPdt) or
        color environment(string name, point direction) or
        float shadow(string name, point position) or
        color texture(string name, float s, t) or
        float texture(string name, float s, t) {
                Texture Operations
            }

        point transform(string fromspace, tospace, point p) {
                Primarily Arithmetic
            }

        point refract(point I, point N, float eta) {
                /* from textbook */
            }

        fresnel(...) {
                /* from textbook */
            }

        color trace(point location, point direction) {
                /* unknown */
            }


@
```